

### **REMARKS**

This response to the Office Action mailed on November 18, 2004 cancels no claims, amends claims 1-5, 31-33, 38, 49, 55-56, and 61, and adds no claims. As a result, claims 1-5, 7-9, 11, and 31-67 remain pending in the Application.

### **Objections to and §112 Rejections of the Claims**

Paragraph 1 of the Office Action alleges mistakes in the claims. The claims are amended to spell “non-essential” in a manner consistent with the Specification. A colon is added to claim 1. Asyndeton usage is made consistent throughout the claims. Applicant would appreciate specific information as to any other informalities.

Claims 2-5 and 31-32 were rejected under 35 USC §112, second paragraph, as indefinite. The “pipelines” of these claims are replaced by the term “memory,” which is found in their parent claims. Claim 32 is amended to recite the “dynamic code analyzer” as a new element.

None of these changes narrow the scope of the amended claims.

### **§102 Rejections of the Claims under Jones**

Claims 1-2, 4-5, 7-9, 11, 31-33, 35-37, 39-60, and 62-67 were rejected under 35 USC § 102(e) as being anticipated by Jones et al. (U.S. 6,745,222). Applicant respectfully traverses these rejections.

The Jones reference concerns a scheduler for multiple programs executing under an operating-system. The scheduler allows real-time programs to meet their time constraints by including a fixed recurring execution-interval length for each such program in a node of a scheduling graph. The purpose of the present invention is quite different: to speed up the execution of an individual program that includes hints and other ancillary code that is non-essential to the essential functional code of the program. From the beginning, then, the invention is not relevant to program scheduling or to time constraints, nor is the reference relevant to any aspect of executing non-essential code.

Independent claim 1 recites certain code as “essential” or “non-essential” code. The Office Action equates essential code to Jones’ real-time code, because “it is essential that the

group of instructions in a real-time thread be executed by a deadline,” and equates non-essential code to Jones’ non-real time code, in that “it is not essential that the non-real time threads are executed by a deadline.” (Office Action, par. 9) This, however, clearly perverts Applicant’s explicitly stated definitions for these terms. For example, the Specification states:

“In this description essential and non-essential code are defined with respect to a particular reference architecture. Essential code is that code which is both necessary and sufficient for correct execution of a program on the reference architecture. Non-essential code is code which is not necessary for correct execution, but which may benefit the performance of the essential code on the reference architecture.” (Page 4, lines 2-6)

It is well settled that a patentee may be his own lexicographer, and that his definitions must be accorded weight. Jones’ non-real time threads are not “non-essential” code. Although they need not be executed within any certain time period, their execution at some time is necessary to the correct execution of Jones’ non-real time programs---these programs simply will not function correctly if their threads are not executed at all. Applicant’s non-essential code, on the other hand, “may be optionally omitted with no impact on the correctness of the executed program.” (Specification, page 4:8-9<sup>1</sup>) Therefore, Jones contains no suggestion of any element corresponding to the “non-essential code” of claim 1.

Claim 1 also stores essential code in a “first memory,” and stores non-essential code in a separately recited “second memory.” The Office Action cites passages in Jones at col. 2:4-24, 33-49 and 21:5-17 to contend that sequences of instructions representing real-time and non-real time threads are stored in different memories. However, the reference itself contains no suggestion that different threads reside in different memories; Jones does not disclose where the threads are stored, or that their manner of storage is of any concern whatsoever. The statement in par. 12 that different threads “have there [*sic*] own space in memory” does not place them in separately loadable and separately configurable memories such as Applicant’s I-cache 104 / main pipeline 106 (for essential code) and H-Flow cache 122 / H-Flow pipeline 120 (non-essential code). Therefore, claim 1 clearly distinguishes Jones in this regard as well.

Claim 1 further declares that the triggers contain “contingent conditions” The conditions are contingent in that they need not necessarily occur during the execution of the essential code. For example, page 9:26 states that the trigger conditions “can be satisfied.” In the description of instruction triggers, page 11:13-15 explains how the a trigger specifying a pointer value is

---

<sup>1</sup> ---The colon notation refers to page and line numbers of the Application, or to column and line numbers of a patent. For example, “page 4:8-9” denotes page 4, lines 8-9 of the Application.

satisfied if the pointer value is encountered in the program. Data, state, and event triggers are described similarly. In Jones' system, the only disclosed trigger for switching from one thread to another are clock times which are not contingent; they always occur.

Claim 1 then recites that the conditions specify predetermined "attributes of the essential code." These attributes are described on page 11:9-13:16 (for instruction triggers), on page 13:19-15:23 (data triggers), on page 15:26-16:2 (event triggers), and on page 16:5-21 (event triggers). Page 16:24-18:20 recounts the use of instruction triggers for virtualization. Jones only possible analog to Applicant's triggers is the expiration of a clock interval, which has nothing to do with any attributes of a code thread. Accordingly, claim 1 manifests at least four clear differences from the Jones reference.

Claims 2, 4-5, 7-9, 11, 31-32, and 55-56 depend from claim 1, and distinguish Jones for the same reasons. For example, claim 4 recites that the conjugate mapping table "is responsive to the microarchitectural state." Par. 11 of the office Action alleges as a pure conclusion that Jones teaches this aspect. In fact, however, Jones scheduler is only responsive to time intervals, with no linkage to any state of Jones' architecture. As to claim 5, the statement in par. 12 of the office Action that both threads are stored in cache "systems" is incorrect; both threads are stored in a single cache system, whereas claim 5 recites two separate caches. As another example, Applicant finds nothing in the passages cited in par. 15 to suggest that the triggers of claim 9 might contain "vector" (i.e., multiple) values. Each node of Jones' scheduler appears to contain at most a single time interval value. Applicant respectfully request a more specific teaching for this assertion in the Office Action. As a further example, claim 31 introduces a dynamic code analyzer to "generate non-essential code from the essential code." Jones only analysis of any code is to determine timing requirements for real-time threads. This analysis does not generate code of any kind whatsoever. Jones' acyclic graph contains nodes each representing a complete thread or program; it does not create a graph containing "trace representations of ... the essential code" as described on page 8:8-11 of the Application.

Independent claim 33 distinguishes the Jones reference for substantially the same reasons as does claim 1. Applicant's clear definition of the term "non-essential code" excludes any interpretation of Jones' non-real time threads. The statement in par. 19 that non-essential code is

stored in a separate second memory is not taught in the cited passages of the reference. Jones' non-real time threads are stored at different addresses of the same memory, and Jones suggests no reason to store anything in a separate memory structure.

Claim 33 further recites that each trigger includes condition(s) "resulting from the execution of the non-essential code." First, Jones has no "non-essential" code. Second, as stated in par. 19 of the Office Action, Jones' "scheduler makes the switch based on the priorities and [timing] constraints, and is predetermined." That is, Jones does not switch based upon any results from his real-time code, any instructions, data, states, or events that a real-time thread may---or may not---encounter during its execution. Instead, Jones makes a switch upon factors that are wholly external to the execution of the real-time code---in fact, external to the execution of all code.

Claims 35-37, 39-48, and 57-67 depend from claim 33, and include other features as well. For example, claims 36 and 37 state that the essential and non-essential code is kept in memories that are "logically separate" and "physically separate." In the cited passage, col. 2:4-24, Jones merely says that a multiple-thread process of either kind, real-time or non-real time, may own blocks of memory (col. 2:18-21); this statement applies to any process, and does not suggest that different types of processes might be segregated, real-time processes being stored in one block as a group while non-real time processes are stored in a different block as a group.

As another example, claim 39 recites that the non-essential code performs "instruction set virtualization." Program scheduling has no relation to virtualization. This term is well defined in the art. For example, "Virtualization at the instruction set architecture level is implemented by emulating an instruction set architecture completely in software." Susanta Nanda et al., "A Survey on Virtualization Technologies," TR179, Dept. of Computer science, SUNY at Stony Brook, at page 6.<sup>2</sup> Jones suggests no virtualization of any kind. Dependent claims 66 and 47 note that the triggers are "state attributes." Page 15:25-16:2 defines state attributes and gives examples. Applicant requests further particularity as to how the cited passages of Jones can be

---

<sup>2</sup> ---Accessed from <http://www.ecsl.cs.sunysb.edu/tr/TR179.pdf>, on March 17, 2005. A definition at SearchStorage.com has the entry "virtualization.... 1) With computer hardware, virtualization is the use of software to emulate hardware or a total computer environment other than the one the software is running in." ([http://searchstorage.techtarget.com/sDefinition/0,,sid5\\_gci499539.00.html](http://searchstorage.techtarget.com/sDefinition/0,,sid5_gci499539.00.html), accessed March 17, 2005)

interpreted to teach “architectural and microarchitectural states (page 15: 27) as a subject for Jones’ scheduler nodes.

Independent claim 49 recites “nonessential” code, whereas all of Jones’ code is “essential” within Applicant’s clear definition noted above. If the code of Jones’ non-real time programs is not executed, then those programs will fail. Moreover, Jones’ non-real time threads or programs are not associated with “the same particular program” as his real-time threads or programs. Every fair reading of Jones suggests to one skilled in the art that real-time and non-real time threads are entirely separate from each other, their only linkage being that they all compete for time in the same processor.

Claim 49 further articulates that the non-essential code “includes code generated from the essential code.” This language follows allowed dependent claim 38. Par. 53 Of the Office Action admits that “Jones, Asghar nor any of the indicate [sic] prior art have not taught individually, or in combination has taught wherein the non-essential code includes hint code generated by a compiler from the essential code.” This statement remains true for any type of code generated from the essential code, regardless of what agency might generate the code. The Office Action equates Applicant’s “essential code” to Jones’ real-time threads, and “non-essential code” to Jones’ non-real-time threads. But all threads of both types are separate entities; none of the real-time threads “generate” any code at all, much less code that may form a part of any non-real time thread. Col. 6:11-27, cited in par. 17 in connection with claim 31, at most shows that Jones places data into nodes of an acyclic scheduler graph. This data structure is not “code.”

Claims 50-54 depend from claim 49, and incorporate all its recitations. In addition, claim 52 stores essential and non-essential code in “separate sections of a file.” Jones does not associate any of his real-time threads with a non-real-time thread<sup>3</sup> for storage together in a single file, and certainly not in separate sections of that file. As to claim 54, Applicant does not find in the cited passage of Jones any reference to a “run-time library.” Applicant derives advantages from this arrangement, especially as to generating non-essential code from the essential code (claim 49), but Jones would not appear to have any reason to incorporate some code in such a

---

<sup>3</sup> ---Even assuming, arguendo, that the non-real time threads could correspond to Applicant’s non-essential code--- which of course they cannot.

library. Applicant therefore requests that the Examiner point out more specifically any correspondence between Jones and this recitation.

### **§102 Rejections of the Claims under Asghar**

Claims 33-34 were rejected under 35 USC § 102(b) as being anticipated by Asghar et al. (U.S. 5,794,068). Applicant respectfully traverses these rejections.

Claim 33 loads the second instruction stream “into a separate memory” from that of the first stream. Par. 6 of the office Action alleges that Asghar’s “data cache holds the operands for the DSP functions and the results, so part of the second instruction steam [sic] must go there to be stored.” Applicant does not understand how this configuration can be inferred. An instruction cache has records or lines each containing a recently-used instruction along with the address where it resides in main memory. When the processor sends out the address of an instruction to be executed, the processor looks in the instruction cache, and, if the address is in the cache, loads the instruction from the cache rather than from main memory. A data cache operates in a similar fashion. Recently-used operands are stored along with their corresponding main-memory addresses. When an instruction requiring operands is executed from the instruction cache or from main memory), the processor looks at the data cache and fetches the operand from the data-cache line having that operand address, if present. That is, a conventional data cache does not store instructions, thus defeating the inference.

Claim 33 further declares that instructions from the non-essential code sequences are “executed in the same microarchitecture structure” as that employed to execute the essential code. The whole purpose of Asghar’s system, however, is to execute a set of normal instructions in a first architecture (GP-CPU Core 212, Fig. 1) and to execute a different set of DSP instructions in an entirely different architecture (DSP Core 214). In a previous response, Applicant demonstrated that the *physical location* of both GP CPU and DSP in the same processor (or chip) is irrelevant to whether or not these two cores have the same *logical architecture*---that is, that they might execute the same instruction set. Par. 6 of the Office Action makes no attempt to rebut this showing, nor does it address the technical reference adduced in support thereof.

Claim 34 inherits the recitations of its parent claim 33, and sets forth other features. Par. 4 of the Office Action cites “figure 4 abstract, [sic] column 4 lines 11-67” as showing two pipelines. Applicant does not find any pipelines suggested in these or other sections of the reference, and respectfully requests a more particular explication for this rejection. Asghar appears to show only cache memories 202 and 444 (Fig. 4).<sup>4</sup>

#### **Allowable Subject Matter**

Claims 3, 38, and 61 were objected to as being dependent upon a rejected base claim, but were indicated to be allowable if rewritten in independent form including all of the limitations of the base claim and any intervening claims. Claims 3, 38, and 61 are so rewritten.

---

<sup>4</sup> --- Cache 444 is not an “instruction” memory structure, as demonstrated in connection with claim 33.

**Conclusion**

For the above and other reasons, the claims meet all statutory requirements, and ought to be allowed. The Examiner is invited to telephone Applicant's attorney at (612) 373-6971 if deemed helpful to facilitate prosecution of this Application.

If necessary, please charge any additional fees or credit overpayment to Deposit Account No. 19-0743.

Respectfully submitted,

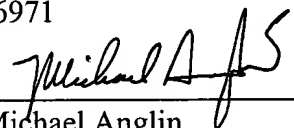
HONG WANG ET AL.

By their Representatives,

SCHWEGMAN, LUNDBERG, WOESSNER & KLUTH, P.A.  
P.O. Box 2938  
Minneapolis, MN 55402  
(612) 373-6971

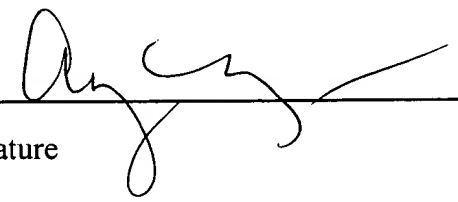
Date 18 Mar 2005

By

  
J. Michael Anglin  
Reg. No. 24,916

**CERTIFICATE UNDER 37 CFR 1.8:** The undersigned hereby certifies that this correspondence is being deposited with the United States Postal Service with sufficient postage as first class mail, in an envelope addressed to: MS Amendment, Commissioner of Patents, P.O. Box 1450, Alexandria, VA 22313-1450, on this 18th day of March, 2005.

Amy Moriarty  
Name

  
Signature